THE UNIVERSITY OF QUEENSLAND

A Tutorial on Induction of Decision Trees

Dr Xue Li
School of Information Technology and Electrical Engineering

INFS4203 / INFS7203 DATA MINING

This tutorial explains a typical data mining algorithm, i.e., ID3 and shows some examples.

Knowledge for Making Decisions (classifications)

There are three ways in which humans acquire knowledge:

- learning by being told
- learning by observation and discovery; and
- learning by induction.

The **knowledge engineer** acquires knowledge from the domain expert by the first two means. These means may be augmented by software, but not automated. This process is very time consuming averaging between 2 and 5 rules per person day! The latter means however may be automated and is a most efficient method of capturing knowledge.

Knowledge acquisition is the activity whereby the machine acquires expert knowledge in order to solve some specific problems. It results in a knowledge base.

Probably, knowledge acquisition is the most difficult aspect in developing an expert system because experts are more likely to *know how* to solve a particular problem under certain circumstances rather than *saying how* to solve problems under various circumstances.

Expert systems need large chunks of knowledge to achieve high levels of performances, and the process to obtain knowledge from experts is tedious and time-consuming; this impasse is universally known as **Feigenbaum bottleneck** (see figure 1). To circumvent this problem, the research believes that: *The only realistic approach to the knowledge acquisition bottleneck is to automate the process (Quinlan,1987).*

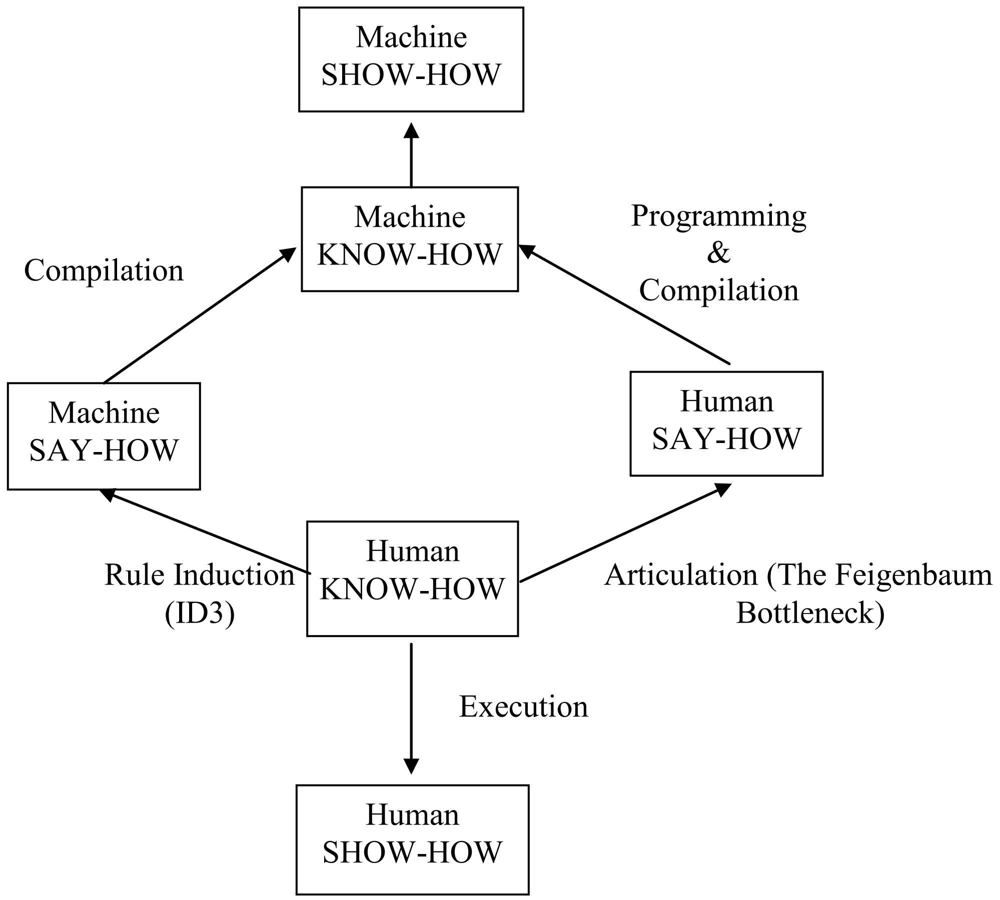


Figure 1. The Map of Knowledge Acquisition.

This process is called **rule induction** (or deductive inference). On the other hand, a conventional way for knowledge acquisition is known as **articulation**.

See figure 1, from *human know-how*, there are two routes to reach the *machine know-how*. One is the articulation aided by knowledge engineers. The other is the rule induction performed by some quite efficient inductive inference algorithms.

A Comparison between Induction and Deduction

In logic terms, the **induction** task is: given Q, find a P such that $P \rightarrow Q$ This is the opposite to **deduction**, where given P and $P \rightarrow Q$ then know Q. Induction is generalizing from the particular, whereas deduction is specializing from the general. Induction is determining the formula from a set of values whereas deduction is finding a value from a formula.

Induction is the process of extracting a pattern from examples.

That pattern may be an integrating concept, or a formula that the values satisfy, or a decision tree that yields an outcome predictable from the values in the examples.

It is this latter aspect that is important in knowledge engineering because values of a restricted set of attributes lead to a conclusion or recommendation.

The induction task has been characterized by Quinlan, Compton, Horn and Lazarus (1987) as :

- given a **training set** consisting of an object described by a fixed set of **attributes**; and
- given an **outcome** known for each object example
- find a **rule** that expresses the objects class in terms of the values of its attributes

We would like to induce a set of rules whose application will yield a recommendation given the values of cost and functionality.

A non-trivial task would be to induce a set of rules that will be applicable in diagnosis from data collected and possibly stored in a database. This has been done for various medical conditions, such as thyroid disorders, kidney disorders etc.

For these cases, a 'relational' table of values is available that record values for various attributes (blood test results, urine test results etc), and an outcome (hyperthyroidic, hypothyroidic, etc). Some tables consisted of 30,000 rows of 50 attributes, and from this table a prediction of outcome based on the attribute values is required.

A specific example:

find a means of predicting which company profiles will lead to a increase or decrease in profits based on the following data:

Profit	**Age**	**Competition**	**Type**
Down	Old	No	Software
Down	Midlife	Yes	Software
Up	Midlife	No	Hardware
Down	Old	No	Hardware
Up	New	No	Hardware
Up	New	No	Software
Up	Midlife	No	Software
Up	New	Yes	Software
Down	Midlife	Yes	Hardware
Down	Old	Yes	Software

This problem has three attributes (Age, Competition, and Type) and one outcome variable (Profit). A simple minded algorithm would be to select the first attribute as the root, the next at the next level etc. till the leaf node becomes the gaol. Such a situation leads to the following rule structure:

If Age = X **and** Competition = Y **and** Type = Z **then** Profit = ?

This structure will yield a branch for each row in the example table, but is not generalizing, not eliminating superfluous attributes, and is unnecessarily complex.

This method is foolish as a full problem space is needed to be generated to cater for unseen problems. It can be shown that the complete classification space for ***m*** attributes is of size:

$$\prod_{j=1}^{m} \mathbf{N_i} \text{ (}N_i\text{ is the number of values of attribute }N_i\text{)}$$

and generating this space and searching the space for what really happens is not sensible. Using this method may also yield empty leaf nodes for cases not yet encountered. Although we may introduce a special value, e.g., '*' to represent a "does-not-care" situation for a attribute to have the size of the space reduced, but this is still a superficial way to reduce the superfluous attributes.

Note that attributes have either truth values or numerical values. The sample space could also include counter example, as well as examples.

A Short Summary:

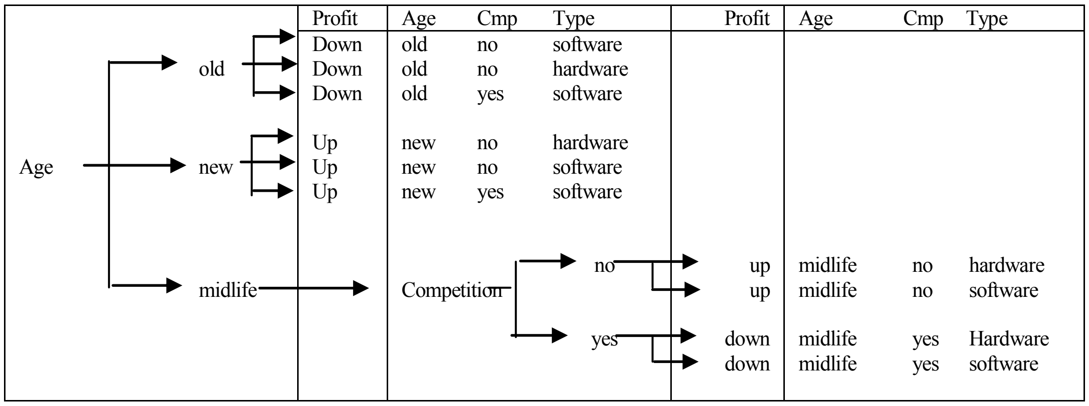
- rules are based solely upon the input examples.
- rules should be able to predict results for other cases.
- if prediction is not achieved, then the rules must be alterable either automatically or editorially.

Building the Decision Tree

- Collect examples to build a test set.
- Decide the class.
- Select one of the attributes to be the starting point or root node of the tree.
- Split up the test set into a number of smaller tables, each containing examples with the same value of the selected attribute.
- If the values in the class is partitioned, then the process is complete. Otherwise, select a new attribute and split the set again.

		Profit	Age	Competition	Type
		down	old	no	software
	old	down	old	no	hardware
		down	old	yes	software
		up	new	no	hardware
Age	new	up	new	no	software
		up	new	yes	software
		down	midlife	yes	software
		up	midlife	no	hardware
	midlife	up	midlife	no	software
		down	midlife	yes	hardware

Tables 1 Split on Attribute Age



Profit	Age	Cmp	Type
Down	old	no	software
Down	old	no	hardware
Down	old	yes	software
Up	new	no	hardware
Up	new	no	software
Up	new	yes	software
up	midlife	no	hardware
up	midlife	no	software
down	midlife	yes	Hardware
down	midlife	yes	software

Tables 2 After the Second Split on Attribute Competition

After the tree is completed, a set of IF-THEN rules from this tree can be produced by following each branch from the root to a terminal node. Each rule is a series of conditions consisting of attribute and value pairs, followed by a single conclusion that contains the class and the corresponding class value.

The intermediate nodes and their branches form the conditions of the rules; the terminal nodes form the rules' conclusions.

R1:	IF age is old	THEN profit is down.
R2:	IF age is new	THEN profit is up.
R3:	IF age is midlife AND competition is no	THEN profit is up.
R4:	IF age is midlife and competition is yes	THEN profit is down.

Question:

What is the best set of rules? Minium, no superfluous predicates etc.

An induction algorithm called ID3 (Iterative Dichotomiser 3) was developed by Ross Quinlan in 1979 and extended in 1983 (i.e., C4.5, see Quinlan 1987). It was designed to produce a decision tree in an efficient manner from large amounts of example data. The material was originally presently in learning **Efficient Classification Procedures and their Application in Chess End Games** 1979 and focused generating a decision tree for the solution of a particular chess problem.

Recall that decision trees can be directly mapped into a production format or conditional expression in a programming language and so can generate the required rules.

ID3 Development:

We want to be able to classify the data present not all-possible permutations of the attribute values. The attributes do interact in some specific manner. We do not what to determine the theory behind the interaction, merely to predict an outcome given a set of valid values.

Thus the problem is not to generate the whole space, just that area actually used.

Secondly, some attributes give stronger predictive capacity than others: e.g., if a person has defaulted on payments frequently before, then the person is a high risk for loans.

How can we identify these attributes, and place these highest on the decision tree? Probability helps here.

Consider computer company data presented before. The understanding of the ID3 algorithm starts from the following question:

> **Which variable AGE, TYPE, or COMPETITION is most strongly associated with PROFIT?**

Note that:

PROFIT has 2 states: UP or DOWN
COMPETITION has 2 states: NO or YES
TYPE has 2 states: SOFTWARE or HARDWARE
AGE has 3 states: OLD, NEW or MIDLIFE

Conditional probability seems most appropriate here to yield a relationship between an outcome state and various attribute values. Note that multi-variate analysis could also be used to show associations, but this technique does not yield decision trees.

Consider the relation between Competition and Profit:

Competition	**Profit**
no	down
no	up
no	down
no	up
no	up
no	up
yes	down
yes	up
yes	down
yes	down

P(Profit= up | Comp = No) = 4/6 = .67
P(Profit= down | Comp = No) = 2/6 = .33
P(Profit= up | Comp = Yes) = 1/4 = .25
P(Profit= down | Comp = Yes) = 3/4 = .75

So what?

Cannot tell any relationship until all either combinations have been calculated.

P(Profit= up | Type = Software) = 0.5
P(Profit= down | Type = Software) = 0.5
P(Profit= up | Type = Hardware) = 0.5
P(Profit= down | Type = Hardware) = 0.5

and for Age

P(Profit= up | Age = Old) = 0.0
P(Profit= down | Age = Old) = 1.0
P(Profit= up | Age = New) = 1.0
P(Profit= down | Age = New) = 0.0
P(Profit= up | Age = Midlife) = 0.5
P(Profit= Down | Age = Midlife) = 0.5

Thus *Age* appears to give the strongest predictive power, and *Type* the least.

Questions:

How can this be formalized? How can this be extended to deeper levels of the decision tree? Need some predictive statistic based on *conditional probability* that can be applied to sort out the 'best' attribute at each level.

Information Measuring

A piece of message can be regarded as an information carrier. A message is not significant by itself. It is significant in the context of all the other possible messages that could have been received. When a message tells you something that you already know, it is reasonably to say that the message conveys no information (i.e., the quantity of information is zero). There was no other possible information. The significance is that the actual message is one selected from a set of possible messages.

The greater the number of possible messages, the greater the amount of information conveyed.

In other words, how much information a message contains depends on the extent to which it resolves uncertainty. From this viewpoint, information is quantifiable in terms of the "number of possible messages".

You may also observe that

The more probable a message is the less information it conveys.

For example, a message selected from a set of only one possible message has a probability of 100 percent, or 1 and conveys no information. A message selected from a set of two equally probable messages, each with a probability of 1/2, conveys some information, while a message from a set of three (probability of 1/3) conveys even more, and so on.

The amount of information increases as the probability of the message decreases; they are inversely related. You could say that the information content of a message with a probability *p1* is 1/*p1*. In the information theory it is believed that:

the total amount conveyed by two messages is equal to the sum of the information conveyed by each of them.

If you have two messages, one with probability of *p1*, and the other with *p2*, you could say that the quantity of information these two messages convey is related to 1/*p1* and 1/*p2*, respectively. However, if you think of the two as a compound message, the probability becomes *p1 x p2*. For example if *p1* is 1/3 and *p2* is 1/5, the probability of two compound messages should be 1/3 x 1/5 = 1/15. Therefore the information content can be quantified by using probabilities such that:

$$I(1/(p1 \times p2) = I(1/p1) + I(1/p2),$$

Where *I* denotes quantity of information. We should now realise that the only mathematical relationship to satisfy this equation is logarithm. Therefore we can say that the quantity of information associated with a probability of *p1* is

$$I(1/p1) = log(1/p1).$$

The choice of the base of the logarithm is based on our intention to reflect the most common situations of the message selection. Base 2 seems a natural choice because, in the simplest case where one of two equally probable messages is selected, each with a probability of ½, the quantity of information is log(1/½), or log2. The log of 2 to the base 2 is 1. Thus by choosing base 2, we are able to deal with the simplest cases and use $\log_2 2$ as a unit of information measuring of 1. (For example, if we choose base 10, $\log_{10}2$=0.30103 would increase the computational complexity in the information theory).

Moreover, when the probabilities of messages are not equal (e.g., *p1* is 1/4 and *p2* is 3/4), the calculation of information quantity for different messages should be averaged. Thus the average information content for *p1* and *p2* of two messages is (1/4 x $\log_2$(1/¼)) + 3/4 x $\log_2$(1/¾)).

In general a "message" could be a specific value appeared in a scientific test, a state of an object, a character in a transmitted string within a communication network, or any data such that its frequency of the appearance in the system is measurable.

One such statistic coming from information theory is called **information entropy** and is a measure of the uncertainty of classification of that object with regards to all objects being classified.

Given a set of objects C and a partitioning c_1 .. c_n, the entropy of this classification scheme is given by (note: Log (1/p) = −log(p)):

$$H(C) = -\sum_{i=1}^{n} p(c_i) * \log_2 (p(c_i)).$$

In the case of classification all examples on the attribute Profit

$$\begin{aligned} H(C) &= -\ (p(\text{Profit} = \text{up}) * \log_2 (p(\text{Profit} = \text{up})) + p(\text{Profit} = \text{down}) * \log_2(p(\text{Profit} = \\ &\qquad \text{down})) \\ &= -\ (0.5 * -1 + 0.5 * -1) \\ &= 1 \end{aligned}$$

This result says that the table is a random collection and by itself has no predictive capacity.

(Your task: Calculate the classification entropies for the other attributes).

We have not focused on the relation of the attribute states to the outcome states. To do so, we must first identify the goal attribute, and measure relevant probabilities for attaining each state of that goal. A measure for each such state can be obtained by using the entropy formula using **conditional probabilities.**

That is

$$\boxed{\mathbf{H(C \mid a_j) = -\sum_{i=1}^{n} p(c_i \mid a_j) * \log_2 (p(c_i \mid a_j))} \qquad \text{(Formula 1)}}$$

Where i = 1 .. n. The function $p(c_i|a_j)$ is the probability that the class value is c_i when the attribute has its *j*th value. More particularly:

$$\begin{aligned} &H(C \mid Comp = no\ ) \\ &\quad = -[p(up|Comp = no)*\log_2(p(up|Comp = no\ )) + p(down|Comp = no\ ) * \log_2(p(down|Comp = no\ ))] \\ &\quad = -[4/6 * \log_2(4/6) + 2/6 * \log_2(2/6)] \\ &\quad = 0.918 \end{aligned}$$

and

$$\begin{aligned} &H(C \mid Comp = yes\ ) \\ &\quad = -[p(up|Comp = yes) * \log_2(p(up|Comp = Yes\ )) + p(down|Comp = Yes) * \log_2(p(down|Comp = Yes))] \\ &\quad = -[1/4 * \log_2(1/4) + 3/4*\log_2(3/4)] \\ &\quad = 0.811 \end{aligned}$$

Similar calculations are made for

H(C| Type = software) = 1
H(C| Type = hardware) = 1

H(C| Age = old) = 0
H(C| Age = new) = 0
H(C| Age = midlife) = 1

How can one aggregate this results to yield a measure for each attribute as a class attribute?

It is simplistic to add the results as in

$$\begin{aligned} &H(C|\ Type\ ) \\ &\quad = H(C|\ Type = software) + H(C|\ Type = hardware) \\ &\quad = 2 \end{aligned}$$

whereas

$$\begin{aligned} H(C|Comp) &= H(C|Comp=Yes) + H(C|Comp=no) \\ &= 0.918 + 0.811 \\ &= 1.729 \end{aligned}$$

etc.

This gives equal weight to all partitions, and does not reflect the manner in which data is distributed. Data distribution can be indicated by weighting using the probabilities of belonging to each class.

Thus

$$H(C|A) = \sum_{j=1}^{m} [\ p(a_j) * H(C|a_j)\ ] \qquad \text{(Formula 2)}$$

Where j = 1 .. m. **m** is the total number of values for the attribute *A*.

That is

$$\begin{aligned} H(C|Comp) &= p(Comp=yes) * H(C|Comp=yes) + \\ &\quad p(Comp=no) * H(C|Comp=no) \\ &= 6/10 * 0.918 + 4/10 * 0.811 \\ &= 0.8752 \end{aligned}$$

Similarly

$$H(C|\ Age) = 0.4$$
$$H(C|\ Type) = 1.0$$

Since Age gives the smallest entropy, and thus the least uncertainly, or greasiest predictive power, the root node is taken to be Age.

Note: Type has an entropy of 1, and is therefore not discriminating, and is not required in a rule! Superfluous attributes can now be identified, and eliminated from the rule set.

The final process is then to select the attribute with smallest entropy and the partition the data on the basis of this attribute's values. Thus the final choice is made from

$$\underset{t=1}{\overset{n}{\text{Minimum}}}\ \{H(C|A_t)\} \qquad \text{(Formula 3)}$$

Where t = 1 .. n. **n** is the total number of attributes for the class *C*.

One observation is that all three formulas of ID3 algorithm can be merged as one expression, since they are nested (i.e., Formula 1 is nested in Formula 2, and Formula 2 is nested in formula 3).

Quinlan actually used an alternative measure based on entropy (i.e., H(C) – H(C|Ai)), but selected the attribute that gave greatest **information gain**. This is equivalent to the attribute with least entropy, and the mathematics is easier.

The method is applied recursively to the data yielding the 'best' attribute at each level. An algorithm is presented later.

The fundamental problem is to select a training set. This set may be offered by an expert as examples. A small set is selected at random, from which a training set, and a verification set is chosen.

Quinlan's ID3 algorithm can then be stated:

- selected a training set;
- repeat
    - induce a rule set to explain the current set
    - find exceptions to this rule set in the remaining examples;
    - form a new rule set from old + existing examples;
- until no more exceptions to the rule (or 'sufficient processing has taken place).

Where the induction method is based on the entropy calculations. Some debate occurs on how to expand the training set on finding exceptions. One method is to accumulate the exceptions in a file to some predetermined size, then add these to the existing training set and redo the induction. This process increases the size of the training set, which increases the computation time to generate the new decision tree.

An alternative is to keep the training set to some specified size and identify key positive examples that must be retained in the training set to generate a meaningful decision tree. Non-key tuples are then swapped for exception tuples when the exception set reached the predetermined size. This alternative keeps the training size constant, and thus keeps the decision tree generation time fixed.

Both methods converge rapidly to a good decision tree capable of discriminating between over 95% of the examples and generally requires but 4 iterations. Only a small size is required in the final training set. The total process time is not strongly dependent on the initial training set size.

ID3 complexity is a linear function of training set size, Complexity of problem as measured by the number of attributes per example .

Note if the training set is too small, then the resultant decision tree has a significant number of empty leaves. If this occurs then the training set needs additional examples.

ID3 can accommodate examples which are designed as positive or negative. ID3 originally allowed only two outcome states, but recent versions can accommodate several states. It was designed to handle large volumes of unstructured example data consisting of example and counter example. ID3 produces **balanced decision trees**.

A problem with ID3 is that the output is not easily understood by experts. The decision tree format is dense, and knowledge not readily accessible by the expert. This problem is reduced if the decision tree is converted into a production system. This is readily done, and commercial systems provide that facility.

ID3 is the most common hub to commercial induction systems. It is extremely efficient in developing a production system from examples and the resultant system is very efficient for extracting answers.

Where is ID3 useful?

When the task is one of classification, such as in diagnosis systems where there exists a database of attributes leading to an outcome. Such systems abound in medicine, fault finding for technical systems, as well as prediction systems such as weather, or in game resolution such as chess, checkers etc.

Note that ID3 has been applied to bridge by Quinlan, as each branch can have a confidence value associated. In all candidates, each object in the problem space must be describable by a fixed number of attributes. The values of the attributes should be contractible to discrete values or a range.

Decision trees are not capable of trapping all knowledge needed in a knowledge-based system. Decision trees are not as rich in representation as first order predicate logic. Nonetheless, many knowledge-based systems can be rendered into production systems.

Problems with ID3

Knowledge presented as a decision tree is dense, and needs transforming into something more accessible to humans.

Noisy data causes decision tree to be of spurious complexity, or too many errors in classification. Noise is introduced because of poor measurement, data entry errors, or inappropriate outcome recorded.

Improved ID3 handles noise by measuring the resultant complexity of the tree on addition of the next attribute, and not permitting the tree to grow unless the predictive power exceeds the complexity.
Unknown attribute values also cause similar problems. Unknown values can be treated as

- special values denoted *;
- replaced with some average for the attribute;
- or by pre-processing the data and not including in the training set examples with unknown values.

the latter yields the best results. If additional examples are not readily available, then pre-processing could replace the unknown values by a random value. This will be treated as noisy data and eliminated by pruning which is described below.

A small training set may result in many empty leaves in the decision tree. The training set needs to be increased in size, and the process repeated. Some training sets have not identified all the relevant attributes. If this occurs, then the resultant decision tree will have identical sub-branches for different class attributes. this is called a clash, and is readily checked by machine and person. The only resolution for this is to call in the expert in order to identify additional attributes.

Advantages of ID3:

- results can be transformed into production rules;
- handles noise
- efficient with large data sets
- easily renders a demonstration prototype on limited data
- eliminates superfluous attributes, possibly ones not known to be poor predictors
- produces balanced decision trees

Dealing with Excessive Branching

Excessive branching occurs with *noisy* and *unknown data*. These problems have been dealt with. **Excessive branching** may also occur when a large number of attributes are included. Not all attributes figure as prominently in predicting the outcome. The relatively unimportant attributes should be discarded.

Quinlan has developed a Pessimistic Pruning Algorithm which replaces a branch with the outcome leave if the errors induced by the replacement is less than one standard deviation of the overall error. This pruning improves the efficiency of the decision tree, and simplifies the decision tree so that the derived rules are more intelligible to human users and domain experts.

Uses of Induction

- overcome the Feigenbaum bottleneck;
- assemble a demonstration prototype from a small example set to show potential of system;
- compressing to essence of rules from an existing rule base;
- converting decision table/tree to production system.

Note: the induced rules are only as good as the data upon which it is based. The resultant system must be verified.

Process of Induction using ID3

- determine main attributes and outcomes of tasks;
- generate or find a training set;
- convert numeric data into discrete values by range;
- apply ID3;
- prune tree.

This process forms the basis of C4 (the first version of C4.5 see Quinland etl 1987), and improved ID3. Its algorithm (from Quinlan Inductive Knowledge Acquisition: A Case Study 1987) is:

```
C4:
- repeat several times:
          GROW:
          - initialise working set:
          - repeat
                    FORM TREE for working set:
                    -         if stopping criterion satisfied,
                              - choose best class
                              otherwise,
                              - choose best attribute test
                              - divide working set accordingly
                              - invoke FORM TREE on subsets
          - test on remainder of training set
          - add some misclassified items to working set
          until no more improvement possible
          PRUNE:
                    While decision tree contains subtrees that are both complex and marginal
                    benefit,
                    - replace subtree by leaf
- select most promising pruned tree
```

Quinlan, Compton, Horn, and Lazarus (1987) claim that C4 is very efficient, requiring generally but 10 iterations to yield the final tree. In its UNIX implementation it is capable of dealing with continuous or discrete data, noisy data and missing attribute values. C4 can recognize infrequent occurrences, and include in the training set as special cases.

The advantage of C4 compared with other algorithms is that it makes its decisions based on data contained exclusively in its training set. Other systems, such as Breiman's on performance on a verification set. This obviously requires more information. C4 also produces a training set that contains informative examples and counter examples similar to that produced by a expert as tutorial examples.

References

Bharath R. (1987): Information Theory, BYTE, Dec 1987, pp291-298

Quinlan, J.R. (1986): Induction of Decision Trees. Machine Learning 1: 81-106. 1986.

Quinlan J.Ross.,etc. (1987): Inductive Knowledge Acquisition: A case Study, in J.R. Quinlan (ed.) Applications of Expert Systems Reading, Addison-Wesley, 1987.

Quinlan J.Ross (1987), C4.5 Programs for Machine Learning, Morgan Kaufmann Publishers, 1987

Thomson B. and Thomppson W., (1986): Finding Rules in Data, BYTE, Nov 1986, pp146-158.